

DETAILED REMARKS

The Office Action (OA) states: "The title of the invention is not descriptive. A new title is required that is clearly indicative of the invention to which the claims are directed."

Applicant respectfully suggests that with the addition of "in a Processor" the title makes it amply clear that the invention deals with the manner instructions are being issued inside a digital processor, namely that the instructions are ordered into queues before their execution.

In order to expand on applicant's contention expressed in "GENERAL REMARKS", applicant would like to briefly overview the present invention's role in the operation of digital processors. The present invention (PI) is dealing with instruction execution by a digital processor. For any processor the executable instruction set gives it its unique characteristics, for instance of an Intel X86 family member, or of a RISC processor, etc. A typical instruction set has a few hundred kind of individual instructions. When an application program is executed by the processor, the program is translated into a stream of instructions which command the activity of the processor. In the prior art this stream of instructions are simply executed in a sequence. The PI is contemplating a more efficient execution of the instruction stream, specifically by exploiting parallel resources available in today's high end processors. It does this by prioritizing the execution of the instructions, and the sorting of the instructions into various queues is a mean toward such an end. That the teaching of the PI is directed in this manner can be found all over the specification, and explicitly spelled out, for instance in the "Summary section", lines 1 through 7 of page 7, and in the language of the claims 1, 14, and 27 claiming apparatus and method for "... issuing instructions..." To illustrate the PI even better, applicant attached to this amendment an appendix page, marked A1, which is copied from the inside back cover of one the standard computer architecture textbooks: John. L. Hennessy and David A. Patterson "Computer Organization and Design" Morgan Kaufmann Publishers, Inc. 1994, ISBN 1-55860-281-X. This page shows the kind of instructions found in the instruction set of an exemplary textbook processor that the authors call MIPS. This is a simplified instruction set in comparison to real system, but it illustrates the subject of the PI. It is the execution of instructions like these that the PI contemplates of classifying, queuing, and logically issuing. It is the decoded binary expression of such entities that are stored in the queues of the PI. Applicant annotated by arrow and underlining the memory related instructions, which here

number two out of the full set of instructions.

Instruction execution is the very heart of processor internal operation, hidden even from application programmers. Of the several hundred kind of individual instructions in a typical processor architecture only a handful deal with memory operations. Such memory operations typically handle ACCESS to the memory in the form of LOAD, (fetching information from the memory), or STORE instructions, (putting information into memory). Applicant emphasizes this because the prior art documents cited by the Office Action (OA) manifestly deal only with effects of memory instructions.

Turning now to the specifics of claim rejections, the OA states: "Claims 1-6, 9, 11, 12, 14-19, 22, 24, 25, and 27 are rejected under 35 U.S.C. 102(e) as being clearly anticipated by Hum et al., US Patent 6,594,730". Applicant recites here that claims 1, 14, and 27 of the PI contain the elements of: "...classification logic adapted for prioritizing instructions...", "...plurality of instruction queues...", and "...issue logic operably coupled to ... instruction queues...". The OA states regarding claim 1 of the PI that Hum et al. teaches: "a classification logic adapted for prioritizing instructions...; a plurality of instruction queues...; an issue logic..." Applicant respectfully submits that Hum et al. teaches no such matters, since the patent 6,594,730 to Hum et al. is directed toward memory controllers, namely how to manage traffic to and from memory, and it says absolutely nothing as regard to instructions. In fact, even the word "instruction" itself does not appear a single time in Hum et al.. Hum et al. states in the "Background" section regarding the scope of their invention in col. 1, lines 6 -12: "The present invention relates to a prefetch queue for use in a memory controller chipset. As is known, modern computer systems may include a memory controller that controls access of other agents, such as microprocessors or peripheral components, to system memory." These statements clearly point out that the scope of Hum et al. is directing access to the memory by other agents, such as a microprocessor. All the while, the contemplations of the PI are solely directed the internal operations of the microprocessor itself. Notice that in the figures of Hum et al., such as 1 and 6, the flow diagrams go to and from the "Memory System", while in all figures of the PI the flow diagrams end in the direction of "Execution".

Hum et al. do teach queues, as various queues can be found in computer science from the earliest of days, but Hum's queues do not contain decoded instructions, but contain memory traffic flow. Everything Hum et al. deals with is the consequence of MEMORY

instructions having been executed in the processor, and Hum et al. is completely silent regarding how, why, or in what order may have such instructions been executed. Also, Ham et al. does not have a "a classification logic" even for its data; note that lines 110, 120, and 130 enter the queues directly, without any previous ordering by specialized logic. Furthermore, although "Arbiters" 170 and 200 in Hum et al. must possess some issue logic, this logic has to deal with memory access, and can't possibly even remotely resemble the "an issue logic operably coupled to said plurality of instruction queues..." of claim 1 and 14 of the PI, which issue logic has to dispatch instructions in an optimized manner to the processor core for execution.

Accordingly, applicant respectfully submits that US Pat. 6,594,730 to Hum et al. teaches in a direction different than the PI, thus it does not anticipate claims 1, 14, and 27, nor does it anticipate the claims depending on 1 and 14, since the dependent claims introduce further limitations.

The OA further states: "Claims 1, 10, 13, 14, 23, and 26 are rejected under 35 U.S.C. 102(b) as being clearly anticipated by Bain, Jr. et al., US- Patent 4,829,425". Applicant respectfully suggests that just as Hum et al., Bain et al. teaches in a completely different direction than the PI. This is clearly declared in the "Summary of the Invention" of Bain et al., col 3 lines 45 -49: "... in accordance with the invention by providing a processor bus sequencer, an I/O bus sequencer, and an execution unit, all of which operate asynchronously and share a common register file memory." There is not even a hint here regarding prioritizing and ordering instructions inside a processor. Bain et al. list the instruction set (of which, referring back to the memory controller of Hum et al., the ACCESS instructions are just a fractional subset) for accomplishing their teaching: col. 11 lines 52 -57, but there is no reference anywhere that there would be any prioritization of these instructions. What Bain et al. are doing is that using this instruction set they are queuing and directing bus traffic, not unlike Hum et al. do with memory traffic. The OA states: "a classification logic adapted for prioritizing instructions in relation to one another and sorting said instructions in a number of priority categories (column 15, lines 30-40, Low priority instructions are classified in the low priority queue and the high priority instructions are classified in the high priority queue.)" In column 15, lines 30-40 Bain et al. described the workings of their Local Bus Sequencer instructions -- and not the queuing of instructions themselves -- even putting the access request queues under

programmer control.

Bain et al. too, do teach queues, as various queues can be found in computer science from the earliest of days, but Bain's high and low priority queues do not contain differing instructions as yet to be executed, but contain results of already executed memory access transactions. Similarly lines 30 - 68 of col 15 describe fairly standard routines, how task scheduling is accomplished by the instructions they are introducing, and not the scheduling of instructions themselves. There is no hint there of an issue logic for instructions. Accordingly applicant respectfully submits that US Pat. 4,829,425 to Bain et al. teaches in a direction different than the PI, thus it does not anticipate claims 1 and 14, nor does it anticipate the claims depending on 1 and 14, since the dependent claims introduce further limitations.

Applicant further suggests that since Hum et al. and Bain et al. teach in different direction than the PI, these cited documents even in combination with Wulf et al., US Patent 6,154,826, and Taniani et al., US Patent 5,655,114, would not render the PI invention obvious to one skilled in arts, since neither Wulf et al., or Taniani et al. has any teachings on issuing instructions.

CLOSING STATEMENT

Applicant submits that, with the amending of the title, this application is now in condition for allowance, which action is respectfully requested.

Respectfully,



George Sai-Halasz, PhD
Registration # 45,430

145 Fernwood Drive
E. Greenwich, RI 02818.
401-885-8032 (Fax 401-885-1046)
E-MAIL - patents@computer.org

Cust. No.: **24299**

MIPS operands

Name	Example	Comments
32 registers	\$0, \$1, \$2, ..., \$31, Hi, Lo	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register \$0 always equals 0. Register \$1 is reserved for the assembler to handle pseudoinstructions and large constants. Hi and Lo are 32-bit registers containing the results of multiply and divide.
2 ³⁰ memory words	Memory[0], Memory[4], ..., Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add \$1,\$2,\$3	\$1 = \$2 + \$3	3 operands; exception possible
	subtract	sub \$1,\$2,\$3	\$1 = \$2 - \$3	3 operands; exception possible
	add immediate	addi \$1,\$2,100	\$1 = \$2 + 100	+ constant; exception possible
	add unsigned	addu \$1,\$2,\$3	\$1 = \$2 + \$3	3 operands; no exceptions
	subtract unsigned	subu \$1,\$2,\$3	\$1 = \$2 - \$3	3 operands; no exceptions
	add imm. unsign.	addiu \$1,\$2,100	\$1 = \$2 + 100	+ constant; no exceptions
	Move fr. copr. reg.	mfc0 \$1,\$epc	\$1 = \$epc	Used to get exception PC
	multiply	mult \$2,\$3	Hi, Lo = \$2 * \$3	64-bit signed product in Hi, Lo
	multiply unsigned	multu \$2,\$3	Hi, Lo = \$2 * \$3	64-bit unsigned product in Hi, Lo
	divide	div \$2,\$3	Lo = \$2 / \$3, Hi = \$2 mod \$3	Lo = quotient, Hi = remainder
	divide unsigned	divu \$2,\$3	Lo = \$2 / \$3, Hi = \$2 mod \$3	Unsigned quotient and remainder
	Move from Hi	mfhi \$1	\$1 = Hi	Used to get copy of Hi
Logical	Move from Lo	mflo \$1	\$1 = Lo	Use to get copy of Lo
	and	and \$1,\$2,\$3	\$1 = \$2 & \$3	3 register operands; logical AND
	or	or \$1,\$2,\$3	\$1 = \$2 \$3	3 register operands; logical OR
	and immediate	andi \$1,\$2,100	\$1 = \$2 & 100	Logical AND register, constant
	or immediate	ori \$1,\$2,100	\$1 = \$2 100	Logical OR register, constant
	shift left logical	sll \$1,\$2,10	\$1 = \$2 << 10	Shift left by constant
Data transfer	shift right logical	srl \$1,\$2,10	\$1 = \$2 >> 10	Shift right by constant
	load word	lw \$1,100(\$2)	\$1 = Memory[\$2+100]	Data from memory to register
	store word	sw \$1,100(\$2)	Memory[\$2+100] = \$1	Data from register to memory
	load upper imm.	lui \$1,100	\$1 = 100 x 2 ¹⁶	Loads constant in upper 16 bits
Conditional branch	branch on equal	beq \$1,\$2,100	If (\$1 == \$2) go to PC+4+100	Equal test; PC relative branch
	branch on not eq.	bne \$1,\$2,100	If (\$1 != \$2) go to PC+4+100	Not equal test; PC relative
	set on less than	slt \$1,\$2,\$3	if (\$2 < \$3) \$1=1; else \$1=0	Compare less than; 2's complement
	set less than imm.	slti \$1,\$2,100	if (\$2 < 100) \$1=1; else \$1=0	Compare < constant; 2's comp.
	set less than uns.	sltu \$1,\$2,\$3	if (\$2 < \$3) \$1=1; else \$1=0	Compare less than; natural number
	set l.t. imm. uns.	sltiu \$1,\$2,100	if (\$2 < 100) \$1=1; else \$1=0	Compare < constant; natural
Unconditional jump	jump	j 10000	go to 10000	Jump to target address
	jump register	jr \$31	go to \$31	For switch, procedure return
	jump and link	jal 10000	\$31 = PC + 4; go to 10000	For procedure call

Main MIPS assembly language instruction set. The floating-point instructions are shown in Figure 4.44 on page 241. Appendix A gives the full MIPS assembly language instruction set.

Name
add
sub
addi
addu
subu
addiu
mfc0
mult
multu
div
divu
mfhi
mflo
and
or
andi
ori
sll
srl
lw
sw
lui
beq
bne
slt
slti
sltu
sltiu
j
jr
jal

Format R
Format I
Format J
Field size

Main MIPS r
opcode (each
destination r
machine lang

A 1